DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

ONDERZOEKSRAPPORT NR 9604

On the Decomposition of Tabular Knowledge Systems

by

Jan VANTHIENEN

Jef WIJSEN

Katholieke Universiteit Leuven

Naamsestraat 69, B-3000 Leuven

ONDERZOEKSRAPPORT NR 9604

On the Decomposition of Tabular Knowledge Systems

by

Jan VANTHIENEN

Jef WIJSEN

D/1996/2376/4

On the Decomposition of Tabular Knowledge Systems

JAN VANTHIENEN
Departement of Applied Economic Sciences
Katholieke Universiteit Leuven
Naamsestraat 69, 3000 Leuven, Belgium
jan.vanthienen@econ.kuleuven.ac.be

JEF WIJSEN
Department of Computer Science
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussel, Belgium
jwijsen@vub.ac.be

Abstract. Recently there has been a growing interest in the decomposition of knowledge based systems and decision tables. Much work in this area has adopted an informal approach. In this paper, we first formalize the notion of decomposition, and then we study some interesting classes of decompositions. The proposed classification can be used to formulate design goals to master the decomposition of large decision tables into smaller components. Importantly, carrying out a decomposition eliminates redundant information from the knowledge base, thereby taking away -- right from the beginning -- a possible source of inconsistency. This, in turn, renders subsequent verification and validation more smoothly.

1. Introduction

The decomposition of knowledge based systems is recognized as an important research issue, e.g. in Kohavi (4), and Moily & Murray (8). In this paper, we look at the decomposition of knowledge bases that consist of decision tables. Decision tables store rules of the form *if **condition(s)** then **action(s)***. They have been successfully applied in the construction, validation & verification and implementation of expert systems. The success of decision tables can probably be attributed to the availability of an intuitive and simple tabular representation. Unfortunately, arbitrarily built decision tables and knowledge bases may grow very large, which, in turn, may compromise our insight in the rules represented. A solution is to decompose decision tables into smaller independent components, which are easier to manage.

Relatively few studies concern the decomposition of decision tables (examples are Hicks (3), Vanthienen & Snoeck (16)). Moreover, some work in this area suffers from a lack of formality. In this paper, we formalize the notion of decomposition in terms of its inverse operator, viz. composition. Not every decomposition is beneficial. Therefore, we define some interesting classes of decompositions that are helpful to the decomposition process.

This paper is organized as follows. The next section introduces some preliminary concepts about decision tables and (verification and validation of) knowledge based systems. Section 3 and 4 concern the composition and the decomposition of decision tables respectively. Section 5 characterizes some important classes of decompositions. Criteria to guide the decomposition decision are elaborated in section 6. Some related work is discussed in section 7. Finally, section 8 contains concluding remarks.

To appear in *New Review of Applied Expert Systems*, 1996

2. Decision Tables, Modularization and Verification & Validation

2.1. Preliminaries

In this section, the notion of decision table is formally defined. We assume the existence of the following pairwise disjoint sets:

- a set ***COND*** of *conditions*,
- a set ***ACT*** of *actions*,
- a set ***VAL*** of *condition values*.

Every condition is associated with a domain of condition values, as follows:

We assume the existence of a total function *Dom* from ***COND*** into 2^{VAL}, the powerset of ***VAL***, such that for every $c \in$ ***COND***, *Dom(c)* contains at least two distinct elements.

Let C be a set of conditions (i.e., $C \subseteq$ ***COND***).
A *condition state over* C is a total function σ from C into ***VAL*** such that for every $c \in C$, $\sigma(c) \in Dom(c)$.

The set of all condition states over C is denoted by C^*.

Example 1. Let $c, d, e \in$ ***COND*** with $Dom(c) = Dom(d) = \{Yes, No\}$ and $Dom(e) = \{Good, Bad\}$.
$\{c{:}Yes, d{:}No, e{:}Good\}$ is a condition state over $\{c, d, e\}$. ■

A *decision table* is a pair $\langle C ; \delta \rangle$, where C is a non-empty set of conditions and δ is a total function from C^* into 2^{ACT}.

Let $T = \langle C ; \delta \rangle$ be a decision table. We write $| T |$ as a syntactic shorthand for C. We write $\| T \|$ for the smallest set of actions containing $\delta(s)$ as a subset whenever $s \in C^*$.

We write I_C for the decision table $\langle C ; \delta \rangle$ with for every $s \in C^*$, $\delta(s) = \varnothing$. I_C is called an *empty decision table*.

Clearly, $| I_C | = C$ and $\| I_C \| = \varnothing$.

Example 2. Let $x, y, z \in$ ***ACT***.
Consider the following function from $\{c, d\}^*$ into 2^{ACT} (call it δ_1):

$$\begin{array}{l} \{\ \{c{:}No,\ d{:}No\} : \varnothing, \\ \ \ \{c{:}No,\ d{:}Yes\} : \varnothing, \\ \ \ \{c{:}Yes,\ d{:}No\} : \{x\}, \\ \ \ \{c{:}Yes,\ d{:}Yes\} : \{y\}\}. \end{array}$$

$\langle \{c, d\} ; \delta_1 \rangle$ is a decision table (call it T_1).
$| T_1 | = \{c, d\}$ and $\| T_1 \| = \{x, y\}$.

Consider the following function from $\{d, e\}^*$ into 2^{ACT} (call it δ_2):

$\{ \{d{:}No,\ e{:}Bad\} : \{x\},$
$\{d{:}No,\ e{:}Good\} : \emptyset,$
$\{d{:}Yes,\ e{:}Bad\} : \{z\},$
$\{d{:}Yes,\ e{:}Good\} : \emptyset \}.$

$\langle \{d, e\} ; \delta_2 \rangle$ is a decision table (call it T_2). ■

In our definition of decision table, we abstract from the graphical representation of decision tables. Two tabular representations of decision table T_1 of example 2 are shown in figure 1. We assume that it is intuitively clear how a decision table can be represented in a tabular format. The abstraction we use suffices to study the notion of decomposition.

c	N	Y	
d	-	N	Y
x	-	X	-
y	-	-	X

d	N		Y	
c	N	Y	N	Y
x	-	X	-	-
y	-	-	-	X

Fig. 1. Two tabular representations of the same decision table.

2.2. The Need for Modularization

Knowledge based system (KBS) development can benefit from modularization for a number of reasons:

Typical approaches in KBS development start from a model that is gradually constructed through interaction with the expert. (Automated) modularization techniques here can provide important feedback to experts and engineers on the overall knowledge structure. In order to *support the modeling phase*, modularization techniques should, therefore, aim at generating a structure that matches real-world constructs as closely as possible.

With respect to *KBS maintenance*, it should be clearly understood that maintainability is one of the factors that will have an impact on modularization decisions. Particularly, it is desirable to bundle components that deal with the same subtopic into the same module, table or packet, since this will reduce the risk of incompleteness and/or creating inconsistencies when changing the knowledge base in respect to that subtopic.

Both in ***verification and validation*** (V&V), modularization can be considered a very important concept. Most research in verification ('building the system right') concentrates on domain-independent techniques such as anomaly detection, aimed at detecting abuse or unusual use of the knowledge representation scheme used (O'Keefe and O'Leary (9)). Validation ('building the right system') often boils down to constructing and evaluating test cases, combined with visual inspection of the knowledge. Since verification algorithms, and extension checks in particular, face a combinatorial explosion as the size of the knowledge base increases, attempts to overcome this problem include partitioning the knowledge base. Although this approach

dramatically reduces the time needed to check each individual module, the possibility of inter-modular anomalies that arise due to dependencies between (components of) different modules is nevertheless not ruled out. It can be easily understood that modularization theory can be of assistance in selecting a partitioning that minimizes the presence of inter-modular dependencies, thereby reducing the need for time-expensive inter-modular checks.

Not only in verification, but also in validation, modularization theory can play an important role, providing a basis for generating an ensemble of test cases with respect to a specific subtopic. In addition, visualization of each of these modules will facilitate direct examination of the knowledge by the expert.

Execution speed may become a critical factor in real world problem solving, due to the growth in inferencing process time as the knowledge base becomes larger. Enhancing efficiency can be performed by modularizing the knowledge base or transforming it into other representations (such as decision tables, as indicated in Colomb & Chung (1)). Furthermore, a combination of both approaches, that is, modularizing a knowledge base into a structure of decision tables might be appropriate in a number of cases.

2.3. Decision Tables and V&V of Knowledge Based Systems

Detection of anomalies, although considered an important part of KBS reliability assurance (O'Keefe and O'Leary (9)), is rarely incorporated into KBS building tools, as indicated in Preece & Shinghal (10). As a result, verification is in practice seldom integrated into the modelling phase, but performed afterwards on the implemented system, by means of a stand-alone verification tool. We however believe that the inclusion of this verification component into an earlier development phase would strongly improve the process of knowledge acquisition and representation, and prevent expensive errors.

Problems of validation and verification have led to the occasional use of schemes, tables or similar techniques in knowledge representation. It has been reported earlier, e.g. in Colomb & Chung (1), Cragun & Steudel (2), Puuronen (11), Vanthienen & Dries (14), that the decision table technique is able to provide for extensive validation and verification assistance. Most of the common validation problems can easily be solved using decision tables, as described in Vanthienen, Mues, Aerts & Wets (15):

- **Consistency and Correctness of Knowledge**

Dividing knowledge over a large number of rules, designed independently, may lead to problems of inconsistency, such as: *Conflict, Cyclical rules, Invalid attribute values, Unreachable conditions.*

- **Non-redundancy of Knowledge**

Redundancy may considerably harm efficiency. The main problem with redundancy, however, is not inefficiency, but maintenance and the risk of creating inconsistencies. Common problems are: *Subsumption, Redundant premises, Redundant rules.*

- **Completeness of Knowledge**

No current system is able to incorporate all knowledge, but within the specific problem area, the following omissions often occur: *Missing knowledge, Unused attribute values or combinations, Unreachable conclusions.*

The definition of a decision table, as proposed in this paper, only allows for single-hit tables, in which every possible case is included in one (completeness criterion) and only one (exclusivity criterion) column. It can easily be understood that the exclusivity criterion is a very important item in the verification of a decision table, since it will enable the prevention of duplicate, subsumed and ambivalent column pairs. The completeness criterion, too, has an important impact on verification, more specifically, where unused inputs are concerned.

Although the use of decision tables has been proposed before in V&V literature, our viewpoint also differs from these other approaches, because we advocate the use of decision tables as a modelling technique on its own, and not merely as a means towards verification of rule-based systems. As pointed out in Preece & Shinghal (10), tools that verify rule-bases after operationalizing them into decision table format, generally fail to find anomalies that stretch beyond simple pairs of rules. In our opinion, using the decision table formalism as a modelling instrument offers significant advantages in verification, because its structured nature eliminates, for a large number of anomaly types, the need for a translation into some other operational form, such as Petri nets, first order logic, etc. (Larsen & Nonfjall (5), Zhang & Nguyen (18), Zlatareva (19), Liu & Dillon (6)), in order to detect them. This makes it possible to integrate an incremental verification component (Meseguer (7)) into the modelling environment itself, without placing a heavy burden on the performance of the workbench used.

3. Composition

In this section, we define a binary operator on decision tables that builds a new decision table from the two argument decision tables. This composition operator will serve as a basis to study the decomposition of decision tables.

Let $T_1 = \langle C_1 ; \delta_1 \rangle$ and $T_2 = \langle C_2 ; \delta_2 \rangle$ be two decision tables.

The *composition of* T_1 *and* T_2, denoted by $T_1 \times T_2$, is the decision table $\langle C ; \delta \rangle$ such that:

1. $C = C_1 \cup C_2$
2. if $c_1 \in C_1^*$ and $c_2 \in C_2^*$ and $c_1 \cup c_2 \in C^*$, then $\delta(c_1 \cup c_2) = \delta_1(c_1) \cup \delta_2(c_2)$.

Example 3. *(continued from previous example)*
Consider the following function from $\{c, d, e\}^*$ into 2^{ACT} (call it δ):

$$
\begin{aligned}
\{\ &\{c{:}No,\ d{:}No\ \ e{:}Bad\} &&: \{x\},\\
&\{c{:}No,\ d{:}No\ \ e{:}Good\} &&: \varnothing,\\
&\{c{:}No,\ d{:}Yes,\ e{:}Bad\} &&: \{z\},\\
&\{c{:}No,\ d{:}Yes,\ e{:}Good\} &&: \varnothing,\\
&\{c{:}Yes,\ d{:}No,\ e{:}Bad\} &&: \{x\},\\
&\{c{:}Yes,\ d{:}No,\ e{:}Good\} &&: \{x\},\\
&\{c{:}Yes,\ d{:}Yes,\ e{:}Bad\} &&: \{y, z\},\\
&\{c{:}Yes,\ d{:}Yes,\ e{:}Good\} &&: \{y\}\}.
\end{aligned}
$$

$\langle \{c, d, e\} ; \delta \rangle$ is a decision table (call it T). It can be verified that $T = T_1 \times T_2$. ■

Lemma 1 lists some interesting properties of the composition operator.

Lemma 1. Let T, T_1 and T_2 be decision tables. Let $C \subseteq |T|$.

$T \times I_C = T = I_C \times T$

$T \times T_1 = T_1 \times T$ *(commutativity)*

$(T \times T_1) \times T_2 = T \times (T_1 \times T_2)$ *(associativity)* ■

4. Decomposition

We now formalize the notion of decomposition of decision tables.

Let T be a decision table. A *decomposition of* T is a set of decision tables $\{T_1, T_2, ..., T_n\}$ with:

1. $n \geq 2$, and
2. $T = T_1 \times T_2 \times ... \times T_n$, and
3. for every $i, j \in \{1, 2, ..., n\}$ such that $i \neq j$, $|T_i| \neq |T_j|$.

The first condition demands that a decomposition contains at least two components. The second condition demands that the original table is the composition of the component decision tables. This means that no information is lost in the decomposition process. The third condition demands that no two component decision tables of a decomposition have all conditions in common.

Let T be a decision table and let $c \in |T|$. Clearly, $\{T, I_{\{c\}}\}$ is a decomposition of T provided that $|T|$ contains at least two elements. This shows that every decision table with at least two conditions, has a decomposition. However, the decomposition $\{T, I_{\{c\}}\}$ is not very interesting from our point of view, because it provides no insight in the structure of T. A decomposition containing an empty decision table is called basic:

A decomposition is called *basic* if it contains an empty decision table; otherwise it is called *nonbasic*.

In the next section, we characterize some interesting classes of decompositions.

5. Characterization of Decompositions

5.1. C_disjoint Decompositions

Decomposition $\{T_1, T_2, ..., T_n\}$ of decision table T is said to be *c_disjoint* iff for every $i, j \in \{1, 2, ..., n\}$ such that $i \neq j$, $|T_i| \cap |T_j| = \emptyset$. That is, no two distinct decision tables of a c_disjoint decomposition have conditions in common.

Example 4.
Consider the following function from $\{c\}^*$ into 2^{ACT} (call it δ_1):

$\{ \{c{:}No\} : \emptyset,$
$\{c{:}Yes\} : \{x\}\}.$

$\langle \{c\} ; \delta_1 \rangle$ is a decision table (call it T_1).

Consider the following function from $\{d\}^*$ into 2^{ACT} (call it δ_2):

$\{ \{d{:}No\} : \emptyset,$
$\{d{:}Yes\} : \{x\}\}.$

$\langle \{d\} ; \delta_2 \rangle$ is a decision table (call it T_2).

Let $T = T_1 \times T_2$. Clearly, $\{T_1, T_2\}$ is a c_disjoint nonbasic decomposition of T. ■

C_disjoint decompositions are interesting. They allow, for example, "executing" component decision tables in parallel without evaluating the same condition twice.

Not every decision table that has a decomposition, also has a c_disjoint decomposition (example 5). Interestingly, if a decision table has a c_disjoint basic decomposition, then the conditions appearing in empty component decision tables are irrelevant for the decision problem at hand (example 6).

Example 5.
Consider the following function from $\{c, d\}^*$ into 2^{ACT} (call it δ):

$$\begin{array}{l}\{\ \{c{:}No,\ \ d{:}No\} : \varnothing,\\ \ \ \{c{:}No,\ \ d{:}Yes\} : \varnothing,\\ \ \ \{c{:}Yes,\ d{:}No\} : \varnothing,\\ \ \ \{c{:}Yes,\ d{:}Yes\} : \{x\}\}.\end{array}$$

$\langle \{c, d\} ; \delta \rangle$ is a decision table (call it T).
Let $\{T_1 , T_2\}$ be a c_disjoint decomposition of T. Without loss of generality, $| T_1 | = \{c\}$ and $| T_2 | = \{d\}$. Clearly, $\| T_1 \|$ and $\| T_2 \|$ must be either $\varnothing$ or $\{x\}$. For T_1 and T_2 , four possibilities can occur. By exploring all possibilities, it can be checked that $T_1 \times T_2$ is always distinct from T. We conclude by contradiction that T has no c_disjoint decomposition. ■

Example 6.
Consider the following function from $\{c, d\}^*$ into 2^{ACT} (call it δ):

$$\begin{array}{l}\{\ \{c{:}No,\ \ d{:}No\} : \varnothing,\\ \ \ \{c{:}No,\ \ d{:}Yes\} : \varnothing,\\ \ \ \{c{:}Yes,\ d{:}No\} : \{x\},\\ \ \ \{c{:}Yes,\ d{:}Yes\} : \{x\}\}.\end{array}$$

$\langle \{c, d\} ; \delta \rangle$ is a decision table (call it T).

Next consider the following function from $\{c\}^*$ into 2^{ACT} (call it δ_1):

$$\begin{array}{l}\{\ \{c{:}No\} \ : \varnothing,\\ \ \ \{c{:}Yes\} \ : \{x\}\}.\end{array}$$

$\langle \{c\} ; \delta_1 \rangle$ is a decision table (call it T_1).

Clearly, $\{T_1 , I_{\{d\}}\}$ is a c_disjoint basic decomposition of T, which shows that condition d is irrelevant for the decision domain at hand. ■

5.2. A_disjoint Decompositions

Decomposition $\{T_1 , T_2 ,..., T_n\}$ of decision table T is said to be *a_disjoint* iff for every $i, j \in \{1, 2,..., n\}$ such that $i \neq j$, $\| T_i \| \cap \| T_j \| = \varnothing$. That is, no two distinct decision tables of an a_disjoint decomposition have actions in common.

Example 7.
Consider the following function from $\{c, d\}^*$ into 2^{ACT} (call it δ_1):

$\{ \{c{:}No,\ d{:}No\} : \varnothing,$
$\{c{:}No,\ d{:}Yes\} : \varnothing,$
$\{c{:}Yes,\ d{:}No\} : \{x\},$
$\{c{:}Yes,\ d{:}Yes\} : \{y\}\}.$

$\langle \{c, d\} ; \delta_1 \rangle$ is a decision table (call it T_1).

Consider the following function from $\{d\}^*$ into 2^{ACT} (call it δ_2):

$\{ \{d{:}No\} : \varnothing,$
$\{d{:}Yes\} : \{z\}\}.$

$\langle \{d\} ; \delta_2 \rangle$ is a decision table (call it T_2).

Let $T = T_1 \times T_2$. Clearly, $\{T_1, T_2\}$ is an a_disjoint nonbasic decomposition of T (however, it is not c_disjoint). ■

A_disjoint decompositions are interesting. For example, if a decision table has an a_disjoint nonbasic decomposition, then it contains actions that are independent of some conditions.

Let T be a decision table and let $c \in |T|$. Clearly, $\{T, I_{\{c\}}\}$ is an a_disjoint basic decomposition of T provided that $|T|$ contains at least two elements. This shows that every decision table with at least two conditions, has an a_disjoint basic decomposition. What about a_disjoint nonbasic decompositions? Clearly, $\|T\|$ must contain at least two elements for T to have an a_disjoint nonbasic decomposition. However, this condition is not sufficient, as shown by example 8.

Example 8.
Consider the following function from $\{c, d\}^*$ into 2^{ACT} (call it δ):

$\{ \{c{:}No,\ d{:}No\} : \varnothing,$
$\{c{:}No,\ d{:}Yes\} : \varnothing,$
$\{c{:}Yes,\ d{:}No\} : \varnothing,$
$\{c{:}Yes,\ d{:}Yes\} : \{x, y\}\}.$

$\langle \{c, d\} ; \delta \rangle$ is a decision table (call it T).
Let $\{T_1, T_2\}$ be an a_disjoint nonbasic decomposition of T. Without loss of generality, $\|T_1\| = \{x\}$ and $\|T_2\| = \{y\}$. From example 5, it is correct to conclude that $\{T_1, T_2\}$ is not c_disjoint. Without loss of generality, $|T_1| = \{c, d\}$ and $|T_2| = \{c\}$. By exploring all possibilities, we see that $T_1 \times T_2$ is always distinct from T. We conclude by contradiction that T has no a_disjoint nonbasic decomposition. ■

5.3. Ca_disjoint Decompositions

Decomposition $\{T_1, T_2, ..., T_n\}$ of decision table T is said to be *ca_disjoint* iff it is both c_disjoint and a_disjoint.

Ca_disjoint decompositions are interesting. If a decision table has a ca_disjoint decomposition, then it contains knowledge about two independent decision domains. Demonstrably, if a decision table has a c_disjoint basic decomposition, then it has a ca_disjoint decomposition.

5.4. A Comprehensive Example

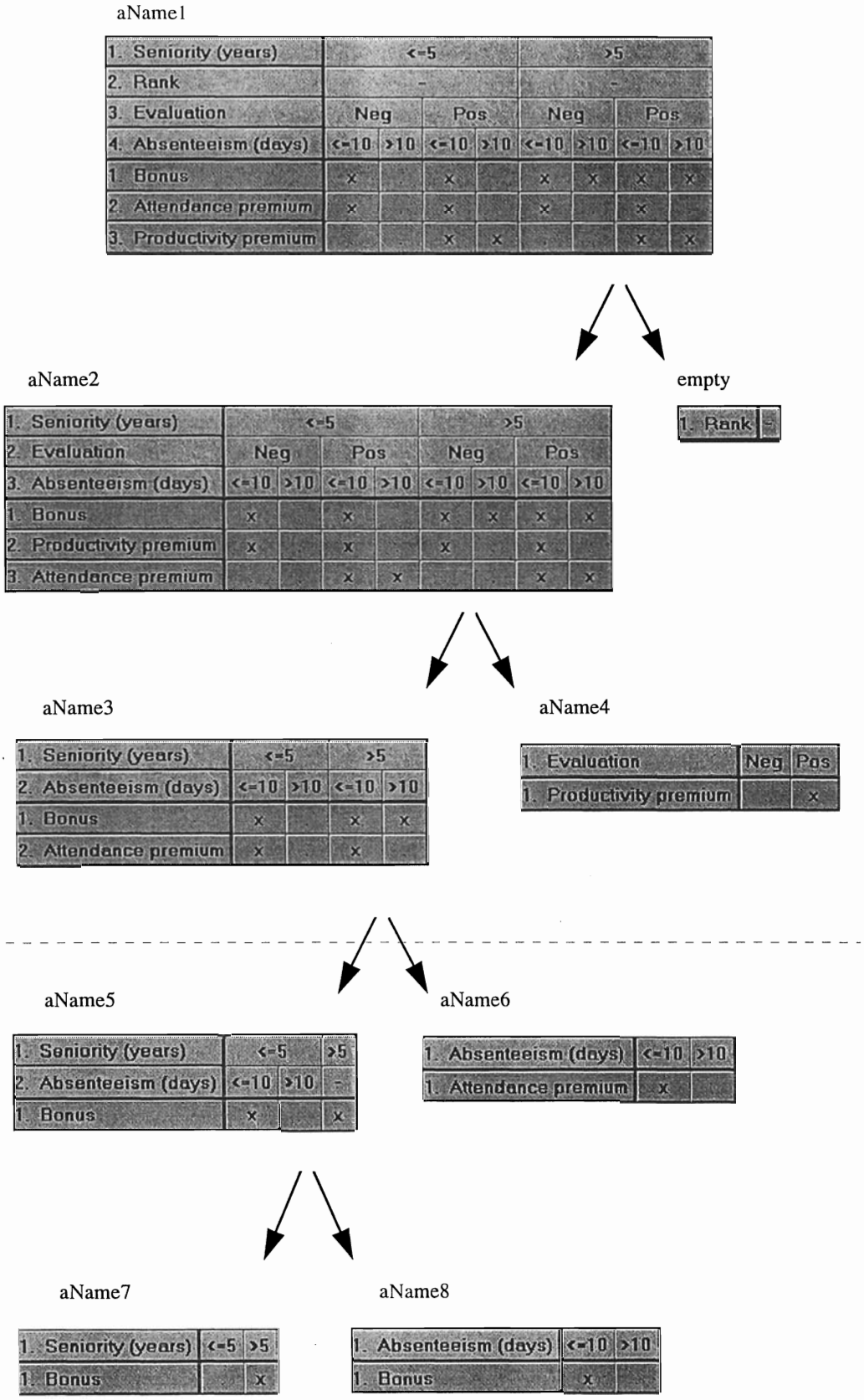

Fig. 2. Example decompositions.

Consider the decision table *aName1* (figure 2). First, a c_disjoint basic decomposition is carried out. This results in, first, the empty decision table $I_{\{Rank\}}$, and second, the decision table *aName2*. The latter decision table can be further "ca_disjoint decomposed", yielding decision tables *aName3* and *aName4* {if *Evaluation* = *Pos* then *Productivity Premium*}. A further a_disjoint decomposition of decision table *aName3* returns decision tables *aName5* and *aName6* {if *Absenteeism* <= *10* then *Attendance premium*}. Finally, decision table *aName5* can be "c_disjoint decomposed" into decision tables *aName7* {if *Seniority* > *5* then *Bonus*} and *aName8* {if *Absenteeism* <= *10* then *Bonus*}. Obviously, subsequent decompositions can be considered a single basic decomposition {$I_{\{Rank\}}$, *aName4, aName6, aName7, aName8*}.

6. Criteria for Decomposition Quality

The proposed classification can be used to set up design goals for the decomposition of large decision tables into smaller components. Importantly, the benefit of a particular decomposition may depend upon the intended use, like validation, consultation, rule generation, etc. For validation purposes, for instance, ultimate decomposition is not always recommendable from a readability point of view (as can be seen in figure 2, where the dashed line indicates the decomposition which is still recommended) :

Ca_disjoint decompositions: Obviously, if a ca_disjoint decomposition is possible, then the original decision table contains unrelated chunks of knowledge. As a consequence, this type of decomposition always results in a simplification of the rules represented.

C_disjoint decompositions: On the other hand, c_disjoint decompositions that are not a_disjoint are not generally recommended. In the above example, the decomposition of *aName5* lowers our insight into the combined influence of *Seniority* and *Absenteeism* on *Bonus*. This, in turn, makes validation more susceptible to errors.

A_disjoint decompositions: It is discussible whether one should always carry out possible a_disjoint nonbasic decompositions that are not c_disjoint. In the above example, the decomposition of *aName3* may not be recommendable, at least not for validation purposes, as it does not reveal under which conditions both a1 and a2 are to be executed. This may be an impediment to validation.

7. Discussion of Related Work

This paper generalizes and formalizes some earlier work on the decomposition of decision tables.

Vanthienen and Snoeck (16) introduce normal forms to "master" the decomposition process. Based on the equivalence between functional dependencies in database design and (a subset of) propositional logic, they indicate how normalization theory can be useful to evaluate a decomposition of decision tables. Although there are major differences between decision table knowledge and database dependencies, the analogy is striking, such that the normalization rules of database design provide an excellent guideline to evaluate the decomposition of decision tables. Both normalization of relations and of decision tables has as primary goal to avoid redundancy

and to correct anomalies. In addition, the normalization of decision tables simplifies decision tables and increases their readability.

Normalization rules for decision tables are then used to investigate how and when a decision table can be split up. Violation of their second normal form, e.g., comes down to the existence of an a_disjoint nonbasic decomposition. Violation of disjunctive second normal form corresponds to the existence of a ca_disjoint decomposition. Finally, violation of partially related second normal form comes down to the existence of an a_disjoint basic decomposition which, however, needs not be c_disjoint. In addition, attention is paid to the decomposition of decision tables into nested decision table structures.

Hicks (3) distinguishes three cases in which simplification of decision tables is recommended. The first case deals with dependencies between conditions. In this paper, we assume that all conditions are independent of each other. The second case concerns the decomposition of a decision table into two decision tables that have no common conditions or actions. In our framework, such a decomposition is categorized as ca_disjoint. Finally, the third case deals with simplifications within a single decision table. The work of Hicks is based on dependencies of the form $X \rightarrow Y$, expressing a kind of functional dependency between conditions and actions. Nevertheless, no formal semantics is given. Especially, the use of negated literals, as in $A \,\&\, \sim B \rightarrow C$, is not explained.

8. Concluding Remarks and Future Research

We gave formal definitions of the decomposition and the composition of decision tables. Furthermore, we characterized some important classes of decompositions. We end with three topics for future research.

An interesting problem that deserves further attention, is the following task: test whether a given decision table has a decomposition of a particular type.

In this paper, we concentrated on "flat" decision tables. Vanthienen and Snoeck (16) suggest that some decompositions may be better represented by *nested* decision tables. In a nested decision table, a reference to a decision table can appear anywhere an action or condition can appear. The transition of flat to nested decision table needs further investigation.

In this study, we made the assumption that conditions are independent of each other. That assumption is relaxed in related work on decision tables and knowledge based systems, e.g. by recognizing impossible condition states, as in Vanthienen, Mues, Aerts & Wets (15). Formalization of these issues, however, is a future research topic.

References

1. Colomb R. and Chung C., Very Fast Decision Table Execution of Propositional Expert Systems, Proceedings AAAI90, 671-676, 1990.
2. Cragun B. & Steudel H., A Decision-Table Based processor for Checking Completeness and Consistency in Rule-Based Expert Systems, *Int. Journal of Man-Machine Studies*, Vol. 5, 1987, 633-648.
3. Hicks R.C.: Minimizing maintenance anomalies in expert systems. *Information and Management*, 28, 1995, 177.

4. Kohavi R., A Third Dimension to Rough Sets, *Third International Workshop on Rough Sets and Soft Computing*, 1994.
5. Larsen H. & Nonfjall H., Modeling in the Design of a KBS Validation System, *Int. Journal of Intelligent Systems*, Vol. 6, 1991, 759-775.
6. Liu, N., Dillon, T., Detecting of Consistency and Completeness in Expert Systems using Numerical Petri Nets, in: Gero, J. and Stanton, R. (Ed.), *Artificial Intelligence Developments and Applications*, North-Holland, 1988, pp. 119-134.
7. Meseguer P., Incremental Verification of Rule-Based Expert Systems, *Proc. 10th European Conference on Artificial Intelligence*, Wiley, 1992, 840-844.
8. Moily J., Murray T.: A modularization approach for Prolog knowledge bases, *Information Systems*, 6, 1993, 405-417.
9. O'Keefe, R. M. and O'Leary, D. E., Expert system verification and validation: a survey and tutorial, *Artificial Intelligence Review*, 7, 3-42, 1993.
10. Preece A. & Shinghal R., Foundation and Application of Knowledge Base Verification, *Int. Journal of Intelligent Systems*, 9, 683-701, 1994.
11. Puuronen S., A Tabular Rule Checking Method, *Proc. Avignon87*, Vol. 1, 1987, 257-268.
12. Rousset, M., Sur La Validité Des Bases de Connaissances: le Système COVADIS, *Proc. Avignon87*, Vol. 1, 1987, pp. 269-282.
13. Vanthienen J. and Dries E., Decision Tables: Refining the Concept and a Proposed Standard, to appear in: *Communications of the ACM*.
14. Vanthienen, J., Dries, E., Illustration of a Decision Table Tool for Specifying and Implementing Knowledge Based Systems, *International Journal on Artificial Intelligence Tools*, Vol. 3, No. 2, 1994, pp. 267-288.
15. Vanthienen, J., Mues, C., Aerts, A., Wets, G., A Modularization Approach to the Verification of Knowledge Based Systems, *Fourteenth International Joint Conference on Artificial Intelligence* (IJCAI 95), *Workshop on Validation & Verification of Knowledge-Based Systems*, Aug. 19, 1995, Montréal, pp. 96-102.
16. Vanthienen J., Snoeck M.: Knowledge factoring using normalization theory, *Internat. Symposium on the Management of Industrial and Corporate Knowledge (ISMICK'93)*, Compiègne, France, 1993.
17. Vanthienen J., Wets G.: Modularization of knowledge based systems. In *Proc. of the Third World Congress on Expert Systems*, 5-9 February, 1996, Seoul, Korea. To appear.
18. Zhang D. & Nguyen D., A Tool for Knowledge Base Verification, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 6, No. 6, Dec. 1994, 983-989.
19. Zlatareva N., A Framework for Verification, Validation, and Refinement of Knowledge Bases: The VVR System, *Int. Journal of Intelligent Systems*, Vol. 9, 1994, 703-737.